EG UK Theory and Practice of Computer Graphics (2013)
Silvester Czanner and Wen Tang (Editors)

Interactive Projective Texturing for Non-Photorealistic Shading of Technical 3D Models

Roland Lux & Matthias Trapp & Amir Semmo & Jürgen Döllner

Computer Graphics System Group, Hasso-Plattner-Institut, University of Potsdam, Germany
{roland.lux | matthias.trapp | amir.semmo | juergen.doellner}@hpi.uni-potsdam.de

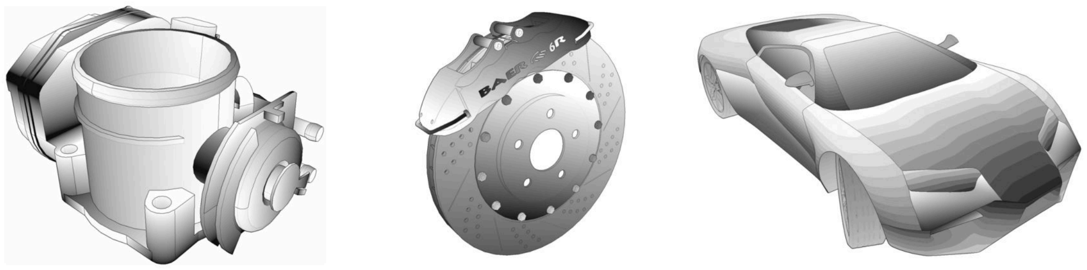


Figure 1: *Exemplary results that have been authored using our system in an interactive way and rendered in real-time.*

Abstract

This paper presents a novel interactive rendering technique for creating and editing shadings for man-made objects in technical 3D visualizations. In contrast to shading approaches that use intensities computed based on surface normals (e.g., Phong, Gooch, Toon shading), the presented approach uses one-dimensional gradient textures, which can be parametrized and interactively manipulated based on per-object bounding volume approximations. The fully hardware-accelerated rendering technique is based on projective texture mapping and customizable intensity transfer functions. A provided performance evaluation shows comparable results to traditional normal-based shading approaches. The work also introduce simple direct-manipulation metaphors that enables interactive user control of the gradient texture alignment and intensity transfer functions.

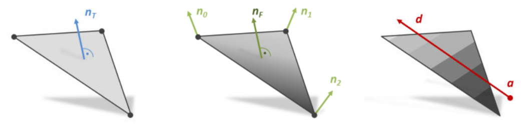
Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Three-Dimensional Graphics and Realism—Display Algorithms Computer Graphics [I.3.6]: Methodology and Techniques—Interaction techniques Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1 Introduction

Three-dimensional geometric models arise in manifold disciplines of information and entertainment systems, in particular they have become essential resources for the architecture, video games, and movie industry. These models often consist of thousands of distinct parts that are manually designed, composed, and iteratively refined using computer-aided design. In this context the application of shading models is essential for visualizing these models to facilitate the perception of shape and to ease the distinction of individual model parts among each other and with respect to the overall structure of the 3D scene.

Visualization techniques based on Gouraud shading [Gou71], Phong shading [Pho75], Gooch shading [GGSC98], and Toon shading variants (e.g., [LMHB00,BTM06]) have proven to be effective approaches for computer-aided design visualization. These shading techniques commonly require surface normal vectors, which represent the input for computing intensity

DOI: 10.2312/LocalChapterEvents.TPCG.TPCG13.101-108



(a) Flat. (b) Phong. (c) Gradient.

Figure 2: *Comparison of normal-based shading approaches and the proposed gradient texturing.*

values based on the respective lighting equation or illumination model. These intensity values are interpolated over the respective surface. Although these approaches significantly improve depth perception, shape recognition, and emphasizes model details, there are cases where the surface details, composition, and orientation of distinct, interconnected model parts are not always clearly exposed to the viewer. Existing techniques for counterbalancing these cases comprise the placement of (additional) light sources [RBD06] or modifications of the surface normals [CST05, MFP11] to exaggerate surface details of an object.

To complement these enhancement approaches on a per-object basis, this paper presents a concept and an implementation of an interactive rendering technique for an alternative variant of non-photorealistic shading (Fig. 1). It is based on a user-controllable shading model that uses directed gradients (Fig. 2) to expose the major orientation and composition of individual model parts. This way, it enables artistic control in creating shading variations, and does not require either light sources or surface normals, nor explicit texture coordinates to be defined. The resulting image styles provide simplifications suitable to communicate virtual 3D models that represent work-in-progress or are meant to be of low detail. Beyond the possibility of artistic modification, our technique can be used for fine tuning or to complement Toon/Cel-shading on a per-object basis similar to [SNS*07]. To summarize, this paper makes the following contributions:

1. It presents a new approach for non-photorealistic shading of technical 3D scenes based on interactive projective texture mapping.
2. It proposes a parameterization of this technique that is suitable for hardware-accelerated rendering.
3. It presents direct-manipulation techniques suitable to modify this parameterization at run-time.

The remainder of this paper is structured as follows. Section 2 reviews related work in the field of technical lighting and shading. Section 3 introduces a rendering concept for 3D models using bounding representations. Section 4 presents interactive manipulation techniques that enable a broad feature palette for artistic direction. Section 5 briefly describes a prototypical implementation of the presented approach using OpenGL and OpenGL Shading Language, and provides a comparative performance evaluation of this implementation. Section 6 discusses application examples, compares the presented approach to existing shading techniques, states problems, limitations, and ideas for future work. Finally, Section 7 concludes this paper.

2 Related Work

Because of the vast body of related work in the area of non-photorealistic rendering, this section focus on the following three categories: *lighting models*, *shading approaches*, and *normal modification* in the context of virtual 3D models.

Gooch et al. present a non-photorealistic lighting model suitable to reproduce technical illustrations by shading only mid-tones, which convey visual prominence of edge lines and highlights [GGSC98]. In [RBD06], a non-photorealistic shading model is presented that is based on dynamically adjusted light positions for different areas of an object's surface. It preserves details independent of the surface orientation and, by operating at multiple scales, simultaneously conveys detail at all frequencies.

Lake et al. presents real-time methods to emulate cartoon styles [LMHB00]. Barla et al. present two real-time capable extensions to this Toon shading approach to support view-dependent effects, such as levels-of-abstraction and depth-of-field [BTM06]. In addition to interpolating between original and abstracted surfaces-normal fields, they utilize a 2D texture to encode how tone varies relative to a given light source and how its detail varies with respect to depth or surface orientation. Kang et al. extend this approach by taking visual saliency into account to depict morphological features of an object more precisely [KCS*09]. They modify local contrast proportional to the degree of visual importance of an area using virtual local lights. Further extensions of toon shading comprise reflective and transparent surfaces [WB02, DE04].

Buchholz et al. present a method to render 3D models using binary shading [BBDA10]. The boundaries between black and white are computed based on minimizing the energy incorporated by the appearance (shading) and geometry (depth and curvature) of a pre-rendered 3D model. In [VVBB11], a concept for the real-time rendering of dynamic stylized shading primitives is presented. It enables artists to modify shading appearance and its dynamic behavior, while supporting a variety of shading methods. With respect to virtual 3D models, Eisemann et al. describe a rendering system that converts a 3D model into stylized 2D filled-region vector-art [EWHS08]. The approach computes region definitions, decomposed into geometric and topological components, that can be stylized with respect to lighting and shading (among others).

In [CST05], a normal enhancement technique for interactive non-photorealistic renderings is presented that improves the perception of an object's shape. In particular, it is well suited to improve the perception of mechanical parts where common shading techniques can of result in shading ambiguities. Miao et al. incorporate visual saliency to adjust the illumination and shading to enhance the geometric salient features of the underlying model by dynamically perturbing the surface

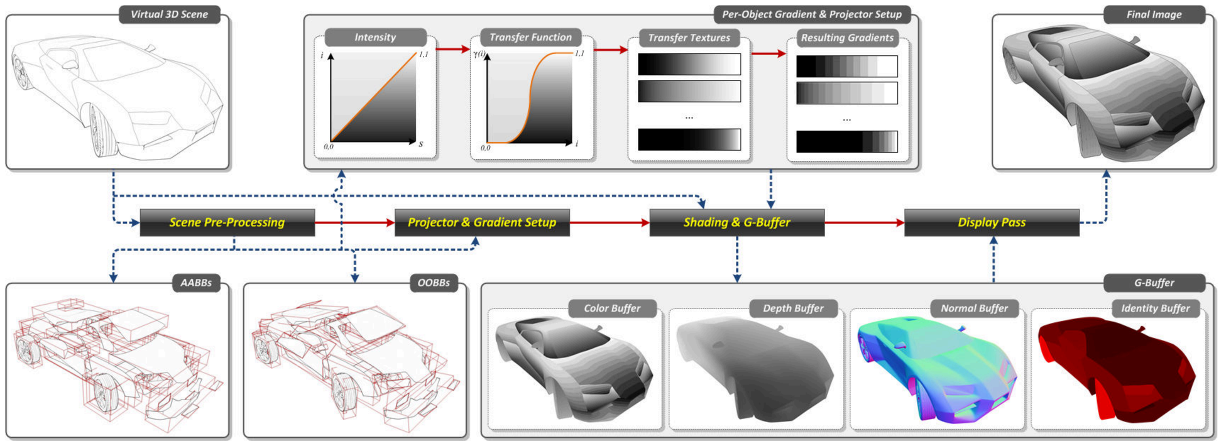


Figure 3: *Schematic overview of our system that implements interactive projective texturing for shading technical 3D models.*

normals [MFP11]. This non-photorealistic shading scheme conveys surface details to improve shape depiction without impairing desired appearance. In this context, the work of [ROTS09] presents a real-time editing system that transforms reflecting surfaces into art-directed reflection surfaces.

3 System and Parameterizations

This section introduces the concept and foundation for rendering a customizable non-photorealistic shading technique based on per-object gradient textures that are placed on the virtual 3D objects. Prior to possible user manipulations, the initial direction and position of the respective gradient textures are estimated using 3D bounding volume approximations of the scene objects.

Preliminaries. It is assumed that a 3D scene ($\mathcal{O}$) is composed of a number of disjunct 3D objects (O), which are composed of polygons (triangles) without requiring a texture parameterization. To texture a respective 3D object using projective texture mapping [Eve01], first, a projector matrix must be composed. Subsequently, the respective texture coordinates are computed for each input vertex.

Given the interpolated fragment coordinates, the final color is computed by synthesizing or sampling the gradient texture using the texture coordinates. Thus, a suitable gradient parameterization is required. Based on such parametrization, adequate interaction techniques for their effective manipulation can be implemented.

System Overview. Figure 3 shows a conceptual overview of the rendering technique that comprises the following three main stages:

Scene Pre-Processing: This pre-processing stage computes 3D bounding volumes for each object in a virtual scene. These bounding volumes are used to compute the initial direction, position, and scale of the respective texture gradient, i.e., the *initial projector setup*. This stage also enriches the object's geometry with unique per-object identifiers that are required for image-based edge enhancement [IFH*03].

Projector & Gradient Setup: Based on the results of the scene pre-processing, heuristics are used to compute initial parameters for projecting the gradients onto the respective objects as well as the gradients itself. The resulting parameters of the projector and the gradient can be manually adjusted and changed at run-time interactively (Section 4).

Interactive Rendering: Finally, the image synthesis is performed by rasterizing each 3D object and applying the respective gradient texture on a per-fragment basis. Here, each object is textured according to its respective *projector* and *gradient settings*. To enhance the resulting rendering, edge-detection and unsharp-masking based on normal, depth, and object-id information is performed in a post-processing pass (Section 5).

The remainder of this section presents details on scene processing as well as the projector and gradient parameterization. For each 3D object with an object identifier id, a feature set F_{id} is defined that describes the respective texturing settings for each object: $F_{id} = (P_{id}, G_{id})$, with $F_{id} \in \mathcal{F}$. Here, P_{id} denotes the *projector parametrization* for computing the texture coordinates using projective texturing mapping, while G_{id} describes the respective *gradient parametrization*.

Scene Pre-Processing. During the scene processing step, the scene graph that represents the virtual 3D model is traversed and each 3D object is processed separately. Each object is represented by its own data structure that stores the geometry, an unique object identifier, and bounding-volume approximations: an *axis-aligned bounding box* (AABB) and an *object-oriented bounding box* (OOBB) [CGM11].

Using these bounding volume approximations, an *anchor point*, a *direction*, and a *scale factor* for the respective gradients are computed using its longest axis for direction and the center point as anchor. In cases where the geometric symme-

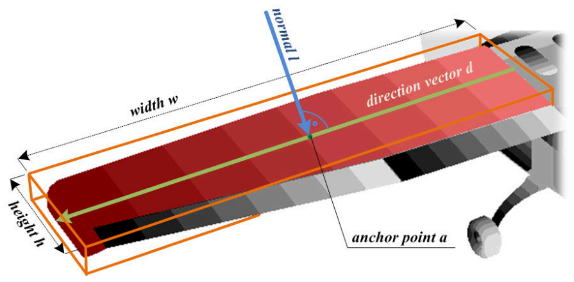


Figure 4: *Projector parameterization based on a bounding volume approximation of the 3D object.*

try of an object should be maintained, the direction vector is chosen to face away from the center of the 3D scene.

Projector Parametrization. Figure 4 (next page) shows an overview of the projector parameterization used by our concept. The setup of a projector $P_{id} = (a, \vec{d}, s)$ with $P_{id} \in \mathcal{P}$ is determined by its anchor point $a = (a_x, a_y, a_z) \in \mathbb{R}^3$ in world-space coordinates, a direction vector $\vec{d} = (d_x, d_y, d_z) \in [-1,1]^3$, and scalar $s \in [0,1]$ that represents the scale or length of a gradient. At run-time, a projector matrix $\mathbf{M}_{id}$ required for projective texture mapping [Eve01] is computed for each of the the projector definitions P_{id} by $\mathbf{M}_{id} := \mathbf{V} \cdot \mathbf{P} \cdot \mathbf{R}$ using the following matrices for the projector orientation ($\mathbf{V}$) and adjustment of the resulting value ranges ($\mathbf{R}$):

$$\mathbf{V} := \begin{bmatrix} b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ h_x & h_y & h_z & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R} := \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, the vector l can be a directional vector perpendicular to d. The half vector corresponds to $h := |l \times d|$ and the binormal to $b := |a - l|$. Finally, the orthographic projection matrix $\mathbf{P}$ is defined as follows:

$$\mathbf{P} := \begin{bmatrix} b_x & b_y & b_z & 1 \\ d_x & d_y & d_z & 1 \\ h_x & h_y & h_z & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given the instance of the projector, the resulting parameter p as input to the gradient function γ is evaluated at runtime for every input fragment with a world-space position V, so that $V' := (p, t, r, w) = \mathbf{M}_{id} \cdot V$.

Gradient Parametrization. The *gradient parametrization* comprises the parameters to map the projected distance p to an intensity value. The following can be generalized to map colors, but for the purposes of the existing implementation an intensity value that is later interpreted as a gray value is sufficient. Each 3D object of a 3D scene can be parametrized according to the following feature set: $G_{id} = (\gamma, \omega)$, with $G_{id} \in \mathcal{G}$.

The gradient parametrization G_{id} is decomposed into an *intensity transfer function* γ and a *gradient function* ω (Fig. 5).

The intensity transfer function is optional and can be used to map a generated texture coordinate p to an intensity value $i := \gamma(p)$. In our prototypical implementation, this mapping function is represented using a parameterized curve. If no intensity transfer is required, the intensity equals the incoming linear distance $i := p$. The final intensity value i' for a fragment position V' is then computed using the gradient function as follows:

$$i' := \omega(i, q, f, p) = (i_q - r) \cdot m_f + (i_q + r) \cdot (1 - m_f) \quad (1)$$

Here, $q \in \mathbb{N}$ defines the *quantization level* of the gradient and $f \in [0,1]$ a *fade-step position* that enables a smooth interpolation between the quantized levels. Further, $p \in [-1,1]$ represents a *power level*, i.e., an exponent x^p, that can be used to non-linearly compress or stretch the intensity. The remaining terms are defined as follows:

$$r := \frac{q+1}{2} \quad i_q := \frac{\lfloor i^p \cdot i \rfloor}{q} + r \quad m_f := \left(i^p \bmod \frac{1}{q} \right) \cdot q$$

This function parameterization enables the synthesis of a variety of gradients (c.f. Fig. 6 for an overview): standard linear and non-linear gradients (Fig. 6(c)), as well as quantizations with hard borders (Fig. 6(b)) or fade-step quantizations (Fig. 6(d)).

4 Interactive Manipulation Techniques

This section describes how a user can interactively manipulate the appearance of the final image. Every parameter of a feature set can be changed interactively without processing the entire 3D scene again. The manipulation possibilities comprise the adjustment of gradient projectors, the intensity transfer function, and gradient function as well.

Object Selection. Prior to the manipulation of the gradient and projector settings, a user can define the respective target objects. Therefore, a user can select or de-select either a *single object* or *multiple objects* (MOS) by double-clicking on the respective items in the 3D scene. For MOS, the current reference object to interact with is depicted using a lighter highlighting color. The reference object of all selected objects can be changed using a single click. To ease the selection process, a user can apply traditional box/circle/lasso selection techniques by stroking closed paths onto the viewing

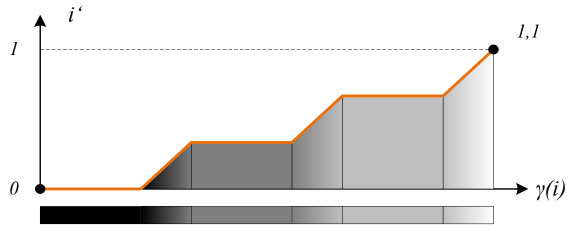
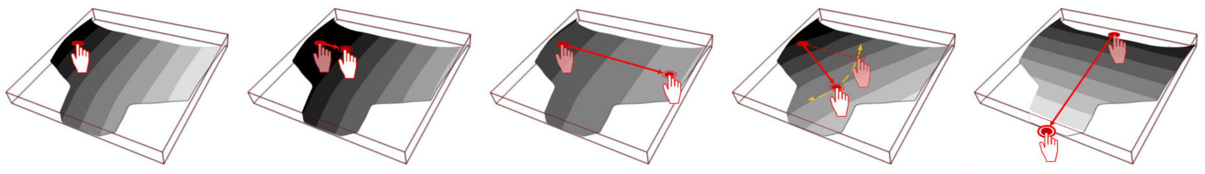
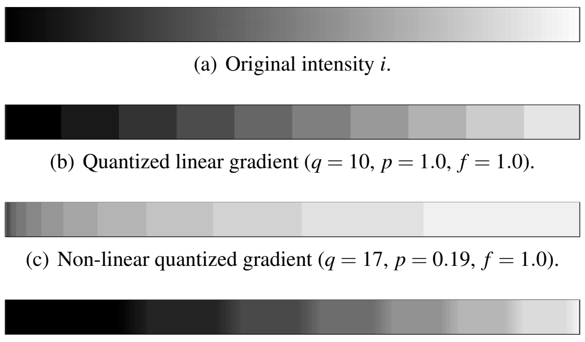


Figure 5: *Exemplary parameterization of the gradient function* ω *with a quantization level* $q = 3$, *a power level* $p = 1.0$, *and a fade-step position* $f = 0.66$.



(a) Initial situation. (b) Anchor manipulation. (c) Scale manipulation. (d) Direction manipulation. (e) Projector manipulation.

Figure 7: *Direct-manipulation metaphors for changing the projector parameters interactively. For simplicity. the images show only a single object part of the virtual 3D model shown in Figure 1.*



(a) Original intensity i.

(b) Quantized linear gradient ($q = 10$, $p = 1.0$, $f = 1.0$).

(c) Non-linear quantized gradient ($q = 17$, $p = 0.19$, $f = 1.0$).

(d) Fade-step gradient ($q = 7$, $p = 1.4$, $f = 0.715$).

Figure 6: *Variants of gradient functions* ω.

plane. An 3D object is selected if the projected center of the bounding volume is inside the polygon described by the path. De-selection can be performed by double-clicking the respective objects again or in the empty canvas areas.

Projector Manipulation. Despite changing the projector settings P using a *widget-based user interface*, the prototypical implementation enables the manipulation using *direct-manipulation metaphors* [Shn87]. Figure 7 shows a conceptual overview of the supported metaphors that can be used with mouse or pen interaction devices, as well as touch interfaces. After a user pressed on a selected scene object (Fig. 7(a)) the following four manipulation modes are available:

Anchor: A user can explicitly change the anchor point a by simply dragging the rendered gradient over the viewport (Fig. 7(b)). Thereby, the anchor point is translated parallel to the reference plane defined by the bounding volume approximation using the offset between the current cursor position and the first hit. A single click centers the anchor point under the cursor. For MOS, the offset is evenly propagated to all selected objects.

Scale: Using this interaction mode, a user can modify the scale s of a gradient without changing its direction or anchor point. Modifying the scale of a gradient results in shorting or widening the gradient (Fig. 7(c)). If MOS is used, the scale value is set for all selected objects.

Direction: This mode enables to manipulate a gradient's direction by rotating the direction vector $\vec{d}$ around the anchor point a (Fig. 7(d)). If MOS is used, the gradient vector of each object is rotated around the same angle using the respective axis l.

Projector: Finally, a user can modify the complete projector setting by simply drawing a line on the viewport (Fig. 7(e)). Analog to the the previous manipulators, it modifies all projector parameters simultaneously, i.e., the anchor point a, the direction $\vec{d}$, and scale factor s.

Gradient Manipulation. The presented system offers a user three possible ways to manipulate the appearance of a gradient setting G. Despite using a 1D image texture, which can be created using standard imaging software, a user can define the gradient procedurally by manipulating each of the parameters individually. This way, a high degree of artistic control is enabled. In addition to modifying the complete intensity function, a user can set a complete 3D object to a chosen gray scale value (including black and white).

To summarize, a user can select between procedural texture gradients and image-based texture gradients. This enables high freedom in the design process, but comes at a cost of higher memory consumptions for complex 3D scenes. The editing functionality was tested with different users. The average time to produce the final images (Fig. 1 and 8) with the presented approaches was between 2-15 minutes.

Algorithm 1 Per-Frame Rendering of Projective Gradients

```
1:  $GBuffer \leftarrow setup()$ [setup G-Buffers]
2:  for all $O_{id} \in \mathcal{O}$ do
3:      $F \leftarrow F_{id} := fetch(id, \mathcal{F})$ [fetch feature set]
4:      $P \leftarrow P_{id} := fetch(id, F, \mathcal{P})$ [projector parameter]
5:      $\mathbf{M} := compose(P)$ [compose projection matrix]
6:      $setupVertexShader(\mathbf{M})$
7:      $G \leftarrow G_{id} := fetch(id, F, \mathcal{G})$ [gradient parameter]
8:      $T \leftarrow fetchIntensityTexture(\gamma)$ [fetch $\gamma$ texture]
9:      $setupFragmentShader(T, G)$
10:     $rasterize(O_{id})$ [rasterize to G-Buffer]
11: end for
12: $GBuffer \leftarrow enhancementPass(GBuffer)$
13: $Framebuffer \leftarrow displayPass(GBuffer)$ [display pass]
```

5 Interactive Implementation

The presented concept was prototypically implemented using OpenGL Shading Language (GLSL) and OpenSceneGraph as rendering engine. Algorithm 1 shows the control flow of the multi-pass rendering technique that comprises the following three passes:

G-Buffer Generation Pass: In the first pass, the complete 3D scene geometry $\mathcal{O}$ is rasterized to create an image-based representation [ST90]. It is performed in a single rendering pass using render-to-texture [Wyn02] in combination with multiple-render-targets. This pass applies projective texturing using the vertex and fragment shader implementations.

Enhancement Pass: This optional post-processing pass applies image-based edge-enhancement [IFH*03] based on discontinuities detected in the normal, depth, and object-id information gathered in the G-Buffer generation pass. It further applies image enhancement by unsharp-masking the depth-buffer [LCD06] to improve depth perception.

Display Pass: Finally, the results of the previous passes are displayed by texturing and rendering a screen-aligned quad that covers the complete viewport. In addition, highlighting operations are applied in this pass.

The first pass implements the gradient diffuse shading by generating texture coordinates using the respective projector matrix $\mathbf{M}_{id}$ at first. The matrix is passed using constant global uniform variables, thus it can change on a per-object basis. Subsequently, the texture coordinate is interpolated by the rasterizer and passed to the fragment shader for evaluation.

```
float computeGradient(
  const in float i, // intensity
  const in float q, // quantization level
  const in float f, // fade steps
  const in float p) // power level
{
 float r = 0.5 / (q + 1.0);
 // apply power-level non-linear transformation
 float ip = pow(i, p);
 // compute quantization
 float iq = (floor(ip * q) / q) + r;
 // compute fade-step intervals
 float mf = mod(ip, 1.0 / q) * q;
 // fade-step interpolation
 mf = step(f, mf) * (mf - f) * 1.0 / (1.0-f);
 // mix between fade steps
 return mix(iq - r, iq  +r, mf);
}
```

Listing 1: *GLSL shader source for computing the procedural gradient defined in Eqn. 1.*

Shader Implementation. In the fragment shader, the interpolated texture coordinates, which are computed using a vertex shader,are passed to a function that represents the gradient parameterization G. All the parameters of the gradient function are passed as uniform variables, so that they can be easily changed at run-time. Listing 1 shows a GLSL implementation of the fragment shader performing the synthesis of the gradient function ω . The customized intensity functions are stored using 1D texture arrays. A texture resolution of 128 pixels for each intensity function is sufficient since distance values can be properly up-sampled using bi-linear filtering.

Comparative Performance Evaluation. The performance of the presented implementation was evaluated and compared with a Phong photorealistic shading approximation [Cla92], as well as two non-photorealistic approaches (Gooch

Table 1: *Geometrical complexity and structural granularity for the test datasets used for the comparative performance analysis.*

Dataset	#Objects	#Vertices	#Faces
Ship	73	12,780	20,505
Car	86	178,922	80,316
Brake	21	32,081	21,246
Aircraft	231	62,727	33,258
Pump	351	6,249	5,807

Table 2: *Results of the comparative performance evaluation of the test datasets described in Table 1 in frame-per-seconds (FPS) without the edge-detection pass.*

Dataset	Gradient	Cel	Gooch	Phong
Ship	40.3	41.6	41.8	41.7
Car	42.9	43.5	42.2	43.9
Brake	45.1	45.7	45.2	44.9
Aircraft	39.2	40.9	40.3	40.1
Pump	37.8	40.3	39.9	40.7

[GGSC98] and Cel-Shading). Table 1 gives an overview for the 3D datasets or scenes of different geometric complexity and structure that were used for evaluation. For the presented approach it is assumed that every object has its distinct feature set assigned, which represents the worst-case scenario.

The performance tests are conducted using a NVIDIA GeForce GTX 460 GPU with 1024 MB video RAM on a Intel Xeon CPU with 2.8 GHz and 6 GB of DDR3 main memory at 532 Mhz. The tests are performed at a viewport resolution of 1600×1200 pixels. The complete model is always visible and no culling is performed. Table 2 shows the results of our performance evaluation. The results do not include the edge-enhancement pass, which decreased the rendering speed at about 2-5 FPS for all of the tested 3D models, regardless of their complexity and the applied shading technique.

The rendering performance of our technique is comparable with other shading techniques. The performance mainly depends on the number of scene objects. For a small number of objects, the four techniques perform similarly. With an increasing scene complexity, our technique is slightly inferior while the local shading model performs almost at the same rendering speed. This can be explained by a more complex setup of the rendering pipeline that is required for our approach. However, the performance of our implementation can compete with the compared shading methods.

6 Results

Applications & Discussion. Figure 8 (next page) shows examples of our shading technique (top row) and compares it with existing shading approaches often used for technical 3D visualizations: the second row shows Cel-Shading, the third

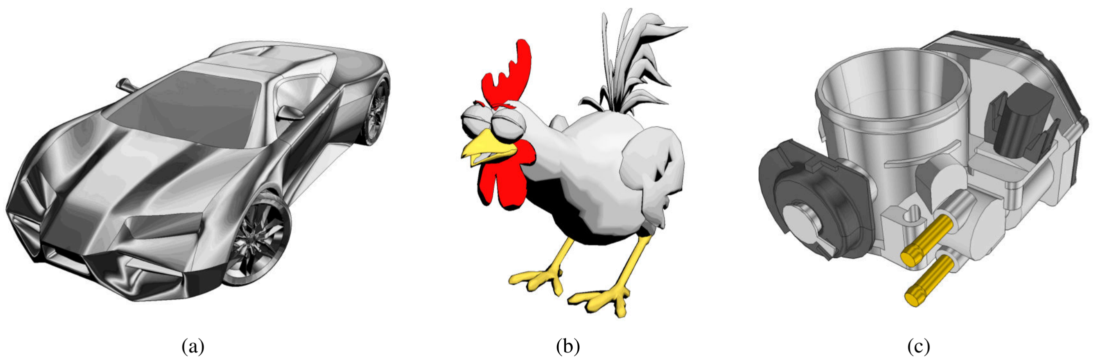


Figure 9: *Using intensity transfer functions γ and gradient functions ω for controlling Toon and Gooch shading on a per-object basis.*

row a variant of Gooch-Shading [GGSC98], and the fourth row depict the results for Phong shading [Pho75] without the specular component. The Cel-shading approach uses a four step color quantization.

As can be seen in the comparison, the quality of our gradient-based diffuse shading technique depends on the granularity of the objects within a 3D scene. It is hardly possible to convey details and surface variations (Fig. 8(c)). On the other hand, our approach enables local intensity changes for objects with large planar areas, which otherwise would have been shaded with the same intensity, i.e., with respect to local shading models that only rely on surface normals (Fig. 8(b) and 8(d)).

However, the presented shading techniques work quite well for manually modeled 3D objects. Since the resulting gradients are independent of light sources and surface normals, and are defined and applied in world-space, they provide frame-to-frame coherence. As a by-product, varying the intensity transfer and gradient function per object enables the modification of Toon and Cel-shading (Fig. 9(a) and 9(b)) by smoothing the intensity quantizations via a fade-step and by facilitating different quantization levels to preserve details. It can also be used to modify the intensity obtained by classical shading methods, e.g., the metal shading described in [GGSC98] (Fig. 9(c)). Finally, quantized gradients could be used to visualize metric information, (e.g., the length of an object or its height by using iso-contours). This would support a human observer to estimate distances, or infer lengths by counting the quantization steps.

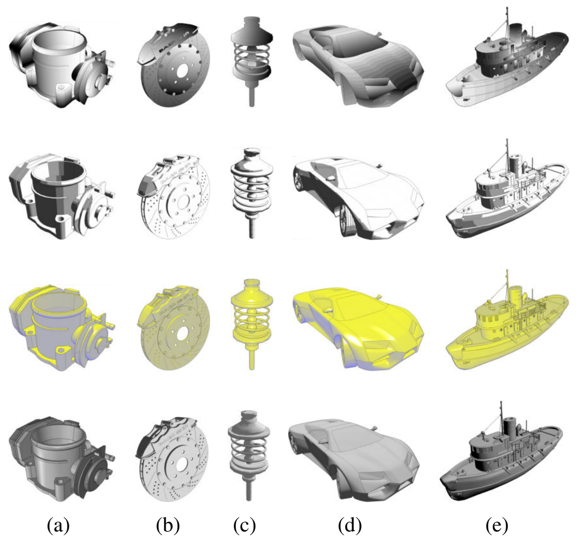


Figure 8: *Comparison between Cel (row 2), Gooch (row 3), Phong-shading (row 4) and the proposed technique (row 1).*

Limitations. Despite the presented heuristics for computing the initial projector and gradient settings, the proposed method requires some manual effort and skills to fine-tune the gradients. The further limitation concerns the object granularity of the virtual 3D scene that depends on various aspects (e.g., the file format used or if batching is enabled). A scene that only comprises a single 3D object would be rendered with only one gradient applied, which kind of defeats the purpose of our technique. On the other hand, a scene that consists of many small objects results in applying gradient textures on a small scale. This can lead to visual clutter, in particular for natural objects (Fig. 10(b) and 10(c)), and virtual 3D scenes with a high number of objects having small surfaces (Fig. 10(d)). Moreover, objects having a complex non-convex or curved surface appear less plastic (Fig. 10(a)). However, the technique is primarily meant for technical illustrations, where the models usually do not suffer from these shortcomings.

Ideas for Future Work. The presented technique has potentials for several future works. Since our approach relies on

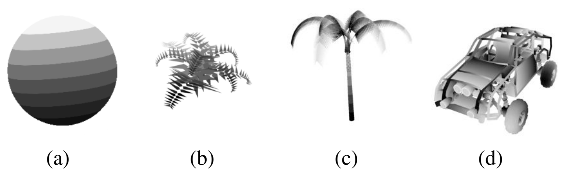


Figure 10: *Exemplary fail cases of the presented shading technique.*

segmented 3D scenes, mesh segmentation algorithms can be applied if no such segmentations are given a priori. Further, a level-of-detail approach (similar to [BTM06]) can be added to improve the rendering quality. Currently, only AABBs and OOBBs are used for gradient orientation. Other (e.g., hierarchical) bounding-volume approximations could improve our technique. For example, bounding spheres [Lar08] include at least two vertices of the geometry it surrounds. These two vertices could be used to define the direction vector. Also the usage of 3D convex hulls for the bounding volume approximation can facilitate the parameterization of the gradient projection. Furthermore , the projector and gradient parametrization can be generalized to consider 2D and 3D textures as well as colored gradients. The developed technique remains to be studied, in particular addressing the reception by domain experts and the ease of use of direct-manipulation metaphors.

7 Conclusions

This paper presents a novel real-time rendering technique that implements interactive projective texturing for non-photorealistic shading of technical 3D models. It is based on projective texture mapping that applies one-dimensional gradients representing the intensity of each scene object and does not rely on surface normals and light sources. The parameters of each mapping can be adjusted at run-time according to the requirements of an artist. The rendering is suitable for technical 3D visualizations and complements the palette of existing non-photorealistic shading techniques. Finally, it can be easily implemented using the texturing mechanisms of graphics hardware.

Acknowledgments

We like to thank the anonymous reviewers for their valuable comments. This work was funded by the Federal Ministry of Education and Research (BMBF), Germany within the InnoProfile Transfer research group "4DnD-Vis" (`www.4dndvis.de`).

References

[BBDA10] Buchholz B., Boubekeur T., DeCarlo D., Alexa M.: Binary Shading Using Geometry and Appearance. *Comput. Graph. Forum 29*, 6 (2010), 1981–1992. 2

[BTM06] Barla P., Thollot J., Markosian L.: X-Toon: An Extended Toon Shader. In *Proc. NPAR* (2006), ACM Press, pp. 127–132. 1, 2, 8

[CGM11] Chang C.-T., Gorissen B., Melchior S.: Fast oriented bounding box optimization on the rotation group $\boldsymbol{SO}(\mathbf{3},\mathbb{R})$. *ACM Trans. Graph. 30*, 5 (2011), 122:1–122:16. 3

[Cla92] Claussen U.: Real time phong shading. In *Proc. EGGH'90* (Aire-la-Ville, Switzerland, Switzerland, 1992), Eurographics Association, pp. 29–37. 6

[CST05] Cignoni P., Scopigno R., Tarini M.: A simple normal enhancement technique for interactive non-photorealistic renderings. *Computers & Graphics 29*, 1 (2005), 125–133. 2

[DE04] Diepstraten J., Ertl T.: Interactive Rendering of Reflective and Transmissive Surfaces in 3D Toon Shading. In *Proceedings of GI Workshop "Methoden und Werkzeuge zukünftiger Computerspiele '04"* (2004), GI, pp. 144–148. 2

[Eve01] Everitt C.: *Projective Texture Mapping*. Tech. rep., NVIDIA Coorporation, April 2001. 3, 4

[EWHS08] Eisemann E., Winnemöller H., Hart J. C., Salesin D.: Stylized Vector Art from 3D Models with Region Support. *Comput. Graph. Forum 27*, 4 (2008), 1199–1207. 2

[GGSC98] Gooch A., Gooch B., Shirley P., Cohen E.: A non-photorealistic lighting model for automatic technical illustration. In *Proc. ACM SIGGRAPH* (1998), ACM Press, pp. 447–452. 1, 2, 6, 7

[Gou71] Gouraud H.: Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers 20*, 6 (1971), 623–629. 1

[IFH*03] Isenberg T., Freudenberg B., Halper N., Schlechtweg S., Strothotte T.: A developer's guide to silhouette algorithms for polygonal models. *IEEE Comput. Graph. Appl. 23*, 4 (July 2003), 28–37. 3, 6

[KCS*09] Kang D., Chung J.-M., Seo S.-H., Choi J.-S., Yoon K.-H.: Detail-Adaptive Toon Shading Using Saliency. In *VIZ '09* (2009), pp. 16–20. 2

[Lar08] Larsson T.: Fast and tight fitting bounding spheres. In *Proceedings of the Annual SIGRAD Conference* (November 2008), Linköping University Electronic Press, pp. 27–30. 8

[LCD06] Luft T., Colditz C., Deussen O.: Image enhancement by unsharp masking the depth buffer. *ACM Trans. Graph. 25*, 3 (July 2006), 1206–1213. 6

[LMHB00] Lake A., Marshall C., Harris M., Blackstein M.: Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering* (New York, NY, USA, 2000), NPAR '00, ACM, pp. 13–20. 1, 2

[MFP11] Miao Y., Feng J., Pajarola R.: Visual saliency guided normal enhancement technique for 3D shape depiction. *Computers & Graphics 35*, 3 (2011), 706–712. 2

[Pho75] Phong B. T.: Illumination for computer generated pictures. *Commun. ACM 18*, 6 (1975), 311–317. 1, 7

[RBD06] Rusinkiewicz S., Burns M., DeCarlo D.: Exaggerated shading for depicting shape and detail. In *ACM Trans. Graph.* (2006), ACM Press, pp. 1199–1205. 2

[ROTS09] Ritschel T., Okabe M., Thormählen T., Seidel H.-P.: Interactive Reflection Editing. *ACM Trans. Graph. 28*, 5 (2009), 129:1–129:7. 3

[Shn87] Shneiderman B.: Direct manipulation: A step beyond programming languages. In *Human-computer interaction*. Morgan Kaufmann Publishers Inc., 1987, pp. 461–467. 5

[SNS*07] Shimotori Y., Nakajima H., Sugisaki E., Maejima A., Morishima S.: Interactive Shade Control for Cartoon Animation. In *ACM SIGGRAPH Posters* (2007), ACM Press. 2

[ST90] Saito T., Takahashi T.: Comprehensible Rendering of 3-D Shapes. In *Proc. ACM SIGGRAPH* (1990), ACM Press, pp. 197–206. 6

[VVBB11] Vanderhaeghe D., Vergne R., Barla P., Baxter W.: Dynamic stylized shading primitives. In *Proc. NPAR* (2011), ACM Press, pp. 99–104. 2

[WB02] Winnemöller H., Bangay S.: Geometric approximations towards free specular comic shading. *Comput. Graph. Forum 21*, 3 (2002), 309–316. 2

[Wyn02] Wynn C.: OpenGL Render-to-Texture. In *GDC Games Developer Conference* (2002), NVIDIA Corporation. 6